

APPLICATION  
FOR  
UNITED STATES LETTERS PATENT

TITLE: DISCOVERY AND INTEGRATION OF JINI SERVICES IN  
NON-JAVA CLIENTS

APPLICANT: KRISHNAMURTHY SRINIVASAN AND EDALA R.  
NARASIMHA

CERTIFICATE OF MAILING BY EXPRESS MAIL

Express Mail Label No. EL688267909US

I hereby certify that this correspondence is being deposited with the United States Postal Service as Express Mail Post Office to Addressee with sufficient postage on the date indicated below and is addressed to the Commissioner for Patents, Washington, D C. 20231.

Date of Deposit November 30, 2000

Signature 

Rich Donovan  
Typed or Printed Name of Person Signing Certificate

DISCOVERY AND INTEGRATION OF JINI SERVICES IN NON-JAVA CLIENTS

BACKGROUND

Jini (TM) is an established specification which allows  
5 application services to announce their availability  
dynamically. Users of services can dynamically find an  
application service using Jini. Examples can include services  
providing freight rates, stock quotes, price inventory levels  
for products or modules (e.g., calculators) for applications.  
10 Jini (TM) operates by registering services, and responding to  
client inquiries about one of these services. Jini (TM) also  
provides proxy code to the client allowing the client to  
communicate with the services. In this way, Jini (TM) acts as  
a broker between the client and the existing services.

15 Jini (TM) has been written in a way such that it can only  
be used by services written in Java. Many non-Java services  
("legacy" services), however exist. These services cannot  
operate through Jini (TM).

20 BRIEF DESCRIPTION OF THE DRAWINGS

These and other aspects will now be described in detail  
with reference to the accompanying drawings.

5

5

5

10

10

20

using Jini (TM), it should be understood that any service requiring a Java API could be similarly used.

The operation is shown in figure 1. A Java service normally publishes itself or its proxy to Jini (TM), along with a set of search attributes that can be used by services that are searching among the services. According to the present system, a non Java service 100 is published, using a Java compliant "wrapper", as described. The wrapper acts like Java proxy code, and points to the non Java service, in the same way that Java proxy code would point to the Java service.

A GUI (Graphical User Interface) allows a user to point and click the location of a service interface file representing the non-Java component, here an ActiveX component. For example, this may be an OCX file for an activeX component 100 as shown in figure 1. The point and click operation is used to instruct the bridge 110 to publish the service.

The bridge 110 operates as shown in the flowchart of figure 2. At 200, the bridge performs run-time introspection of the service component 100. At 210, the bridge identifies the methods/functionality within the service. This can use dynamic inspection and/or keywords, metatags, or other kinds of application inspection. At 220, these results are stated as a list of different functionalities and other relevant

information. Figure 1 shows the functionalities graphically as 120. This list will be used to form the eventual code.

At 230, this system forms middle tier tunneling proxy code 140 to form a bridge between the client and the service.

5 Other information obtained from introspection at 210 is used to generate search attributes. These form keywords which are used to supplement the keyword repository in Jini (TM). In addition to the keywords identified by introspection, a user may also specify additional keywords in the Graphical User  
10 Interface (GUI).

Jini (TM) stores the proxy objects and a set of search attribute objects shown as 152, 154, shown in figure 1.

The Jini (TM) service is shown as 150. A client, or a service-proxy for the client, makes a call to the backend  
15 service wrapper object. The wrapper object redirects the call to the actual Component Object Model ("COM") or CORBA component. Each time such a call is made, the bridge 110 generates code that redirects the calls to the service 100. The generated code may be Java proxy code.

20 At 260, the bridge receives a Jini (TM) lease object based on the successful registration of the Jini (TM) object. The lease object shown as 142 keeps the object definition up-to-date. The bridge renews the lease from time to time, thereby insuring up-to-date information in Jini (TM).

A service is published to Jini (TM) in the following way. First, the service is serialized as a Java object, if it can be so serialized. Other, non Java compliant services are packaged as described above. Services which are incapable of being serialized into Jini (TM) may publish their proxies instead. Constraints such as heaviness, native dependability, and the like may prevent the service from directly publishing to Jini (TM). The proxy is a serializable lightweight Java object that acts as the service delegate. The proxies act as middle tiered objects allowing access of services on the back end. The object or proxy is sent to Jini (TM), along with a set of search attributes. Once the Java objects have been transported to Jini (TM), they are delivered to clients responsive to notifications of service matches.

Another aspect is to allow publishing either Jini or non Jini services. Any service that is capable of serialized in itself, and publishing itself to Jini (TM), would do so directly. This enables clients to directly use the service objects.

Non Jini (TM) services may not have a proxy, or such a proxy may need to be generated as part of the Jini (TM) registration process.

Certain distributed application environments allow services to register with them through an identifier. One example is a GUID key for COM components.

As an example of operation, figure 3 shows how the system could publish and obtain different information. A service 300, e.g., a CORBA service is coupled to the code generator 305 which provides an wrapper around the CORBA shown as 310. Once in the wrapper, the CORBA code appears to the broker or proxy as Java code, i.e., it is no different like any other Java application. At 315, the application is published with the Jini (TM) broker. Figure 3B, shows a client, which is a non Java client such as an Excel(TM) client asking for services. At 330, the request for services is also placed in a wrapper at 335 and placed to Jini (TM). Jini (TM) returns the request.

Although only a few embodiments have been disclosed in detail above, other modifications are possible.